Computer Science & Engineering Department
I. I. T. Kharagpur

Operating System: CS33007
3rd Year CSE: 5th Semester (*Autumn 2006 - 2007*)
Lecture IV (Linux System Calls III)

Goutam Biswas *Date:* 7th August, 2006

1 Communication through pipe

1. A *pipe* (unnamed[1]) allows data transfer from one process to another *related process* in FIFO style. It also takes care of synchronization i.e. the reading process waits if the pipe is empty etc.

```
/*********************************************
 * This program creates and uses an unnaned   *
 * pipe: pipeProg1.c                         *
 * ******************************************/

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

int fact(int n) ;

int main() {
      int chPID1, chPID2, fd[2], err, status ;

      err = pipe(fd) ;
      if(err == -1) {
            printf("Problem in pipe open\n") ;
            _exit(status) ;
      }
      if((chPID1 = fork()) != 0) { // Parent process after child 1
            if((chPID2 = fork()) != 0) { // Parent process after child 2
                  printf("Enter a positive integer:\n") ;
                  waitpid(chPID1, &status, 0) ;
                  waitpid(chPID2, &status, 0) ;
            }
            else { // 2nd Child process
                  int n ;
```

[1] There is a special file called *named pipe*.

```
                close(1) ;  // STDOUT closed
                dup(fd[1]) ;
                close(fd[1]) ;
                scanf("%d", &n) ;
                printf("\t\t\t%d --> Child 2\n", n) ;
            }
      }
      else { // 1st Child process
            int n ;

            close(0) ;  // STDIN closed
            dup(fd[0]) ;
            close(fd[0]) ;
            scanf("%d", &n) ;
            printf("\t\t Child I --> %d! = %d\n", n, fact(n)) ;
      }
      return 0 ;
}

int fact(int n) {
      if(n == 0) return 1 ;
      return n*fact(n-1) ;
}
```

2. Here is another program that shows communication among a parent and two children:

```
/***********************************************
 * This program creates and uses an unnaned   *
 * pipe: pipeProg2.c                          *
 * *********************************************/

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

int fact(int n) ;

int main() {
      int chPID1, chPID2, pCh1[2], ch1Ch2[2], ch2P[2], err, status ;
      int data, copy, sum = 0 ;

      printf("Enter a +ve integer: ") ;
      scanf("%d", &data) ; copy = data ;

      err = pipe(pCh1) ;
      if(err == -1) {
```

```
        printf("Problem in pipe open: pCh1\n") ;
        _exit(status) ;
    }
    err = pipe(ch1Ch2) ;
    if(err == -1) {
        printf("Problem in pipe open: ch1Ch2\n") ;
        _exit(status) ;
    }
    err = pipe(ch2P) ;
    if(err == -1) {
        printf("Problem in pipe open: ch2P\n") ;
        _exit(status) ;
    }

    if((chPID1 = fork()) != 0) { // Parent process after child 1
        if((chPID2 = fork()) != 0) { // Parent process after child 2
            dup2(1, 9) ; // Dup STDOUT to 9

            close(1) ;   // STDOUT close
            dup(pCh1[1]) ; close(pCh1[1]) ; // P -> Ch1
            close(0) ;   // STDIN close
            dup(ch2P[0]) ; close(ch2P[0]) ; // Ch2 -> P
            while(data){
                printf("%d %d\n", data-1, sum+data) ;
                scanf("%d%d", &data, &sum) ;
            }
            close(0); close(1) ;

            dup2(9, 1) ;     // STDOUT
            printf("Sum 1+ ... + %d = %d\n", copy, sum) ;

            waitpid(chPID1, &status, 0) ;
            waitpid(chPID2, &status, 0) ;
        }
        else { // 2nd Child process
            close(1) ; // STDOUT close
            dup(ch2P[1]) ; close(ch2P[1]) ; // Ch2 -> P
            close(0) ; // STDIN close
            dup(ch1Ch2[0]) ; close(ch1Ch2[0]) ; // Ch1 -> Ch2
            scanf("%d%d", &data, &sum) ;
            while(data){
                printf("%d %d\n", data-1, sum+data) ;
                scanf("%d%d", &data, &sum) ;
            }
            printf("%d %d\n", data, sum) ;
            close(0); close(1) ;
        }
```

```
    }
    else { // 1st Child process
                close(1) ; // STDOUT close
                dup(ch1Ch2[1]) ; close(ch1Ch2[1]) ; // Ch1 -> Ch2
                close(0) ; // STDIN close
                dup(pCh1[0]) ; close(pCh1[0]) ;     // P -> Ch1
                scanf("%d%d", &data, &sum) ;
                while(data){
                      printf("%d %d\n", data-1, sum+data) ;
                      scanf("%d%d", &data, &sum) ;
                }
                printf("%d %d\n", data, sum) ;
                close(0); close(1) ;
    }
    return 0 ;
}
```

3. The system call `makefifo` creates a named pipe.

```
/**************************************************
 * This program takes two arguments, creates       *
 * a named FIFO, a reader process (r) and a writer  *
 * process (w): namedPipeProg.c                    *
 * $ ./a.out r nmdpipe &                           *
 * $ ./a.out w namdpipe &                          *
 * ***********************************************/

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define MAX 100
int main(int count, char *vect[]) {
     int err, pd ;
     char wBuff[] = "--> This text will be written in pipe",
          rBuff[MAX] = {0};

     if(count < 3) {
          printf("Less number of arguments\n") ;
          exit(0) ;
     }
     err = mkfifo(vect[2], 0666) ;
```

```
        if(err !=0 && errno == EEXIST) printf("File exists\n") ;
        if(strcmp(vect[1], "r") == 0) { // Reader process
             pd = open(vect[2], O_RDONLY) ;
             read(pd, rBuff, MAX);
             printf("String: %s\n", rBuff) ;
        }
        else if(strcmp(vect[1], "w") == 0) { // Writer process
             pd = open(vect[2], O_WRONLY) ;
             write(pd, wBuff, strlen(wBuff)) ;
        } else {
             printf("Wrong 2nd argument\n") ;
             exit(0) ;
        }
        return 0 ;
   }
```

References

[1]

[2]